Report on the NASA FFT Project

Feasibility study, Software Design, Layout and Simulation of a Two-Dimensional Fast Fourier Transform Machine for use in Optical Array interferometry

Valentín Boriakoff, Wei Chen

P-74

Worcester Polytechnic Institute

July, 1990

NASA Grant Number NAG 5 1138

Principal Investigator: Prof. Valentín Boriakoff

(NASA-CR-186922) FEASIBILITY STUDY, SOFTWARE DESIGN, LAYOUT AND SIMULATION OF A TWO-DIMENSIONAL FAST FOURIER TRANSFORM MACHINE FOR USE IN OPTICAL ARRAY INTERFEROMETRY (Worcester Polytechnic

N91-13885

G3/60 Unclas 0302489

Contents

List of Figures

List of Tables

1 Introduction

The NASA-Cornell University-Worcester Polytechnic Institute FFT chip based on the architecture of the systolic FFT computation as presented in the paper "FFT Computation with Systolic Arrays, A New Architecture," by V.Boriakoff [BOR88] is implemented into an operating device design through the agency of the NASA Grant NAG 5-1138. The kernel of the system, a systolic inner-product floating point processor, was designed to be assembled into a systolic network that would take incoming data streams in pipeline fashion and provide an FFT output at the same rate, word by word. It has been thoroughly simulated for proper operation, and it has passed a comprehensive set of tests showing no operational errors. The chip is labeled SAFT64 (Systolic Array FFT – 64 point).

The following are the "black box" specifications of the chip, they conform to the initial requirements of the design as specified by NASA.

Chip technology: CMOS 2micron ($\lambda = 1\mu$m).

Chip embodiment: 132 pin ceramic package, to be fabricated through MOSIS.

Chip area: 7.9mm×9.2mm.

Power supply voltage: 5.0v.

Input and output data format: IEEE Standard for Binary Floating-Point

Arithmetic (ANSI/IEEE Standard 754-1985), 32 bits.

Internal format: Changed from IEEE standard for computational purposes, returned to IEEE standard at the chip output.

Input clock: 250 ns period with duty cycle from 30% to 70%. Simulation showed proper operation for clock periods from 190ns to 400ns.

Input Reset: must be applied at least four clock cycles before the first valid data word arrived at the input and has to be held high until after the first clock positive rise. It should be dropped before the second positive rise of the clock.

Input data timing: data must be present and valid 4ns before the corresponding clock rise.

Input data sequence: the first data word A_r present on the input data lines must be the real part of the input data word, and the second data word A_i must be the imaginary part of the input data word. A_r should be present 4ns before the first clock positive-going transition, A_i must be present 4ns before the second clock positive transition, 250ns after the first clock positive transition. The C_r input must be applied 4ns before the next positive clock transition (500ns), and C_i must follow C_r 4ns before the following positive clock transition (750ns). All of these data words must be present for one clock cycle. New A_r and A_i values should follow C_i immediately. After the

input data sequence (64 complex words) is complete a new sequence can start at the next positive-going clock transition.

Output sequence: three positive clock transitions after the transition where the input A_r is applied A_r appears at the output without modifiction. Here we count the first clock transition as the one that took A_r into the chip. After A_r appears, it is followed by A_i one clock period later. Yet one clock period later the result of the computation real number R appears at the output. One clock later I is presented at the output. See the timing diagrams.

Step size: Specifies the size of the step of the W coefficients in the ROM, it is a variable that depends on the location of the particular chip in the processor array. See the text for setting values.

Diagonal specification: Two input lines LOWER and MAIN specify which matrix diagonal is being computed in the multiplication. If the value of LOWER and MAIN are 11, the upper diagonal is selected; 01 means main diagonal; 10 means lower diagonal; and 00 is undefined.

Number of inner-product processors required: Based on the initial paper the number of processors required is $P = 3\log_2(N)$, where N is the (binary) number of points in the transform, in this case 64, hence $P = 18$.

Total number of external memory locations required: The data requires $M = 2N - 4\log_2(N) + 2$ memory locations (from the initial paper), hence

$M = 106$. An additional number of N one-bit memory locations are required for the Reset signal.

In addition to our work, initial work was carried out by a Cornell University graduate student (now graduated), Peter DelVecchio, whose collaboration we gratefully acknowledge. His PhD thesis work consisted in the development of hardware verification by software methods, he applied these methods to the verification of those sections he designed for this project.

In the following sections, section 2 describes the five subcells. Their high level function description, logic diagrams and simulation results are presented. Section 3 deals with modification of the design. Since some errors have been found in the ROM, some correction were made. At the mean time, the original design would not be changed. Section 4 discusses simulation methods. Because a four-stage pipeline structure has been used, simulating such a structure is more difficult than an ordinary structure. Section 5 explains chip signal protocols and chip pinout. and Section 6 presents a concrete example of how to utilize the SAFT64 array processors to implement a 64-point FFT. All top level simulation results are included in the Appendix.

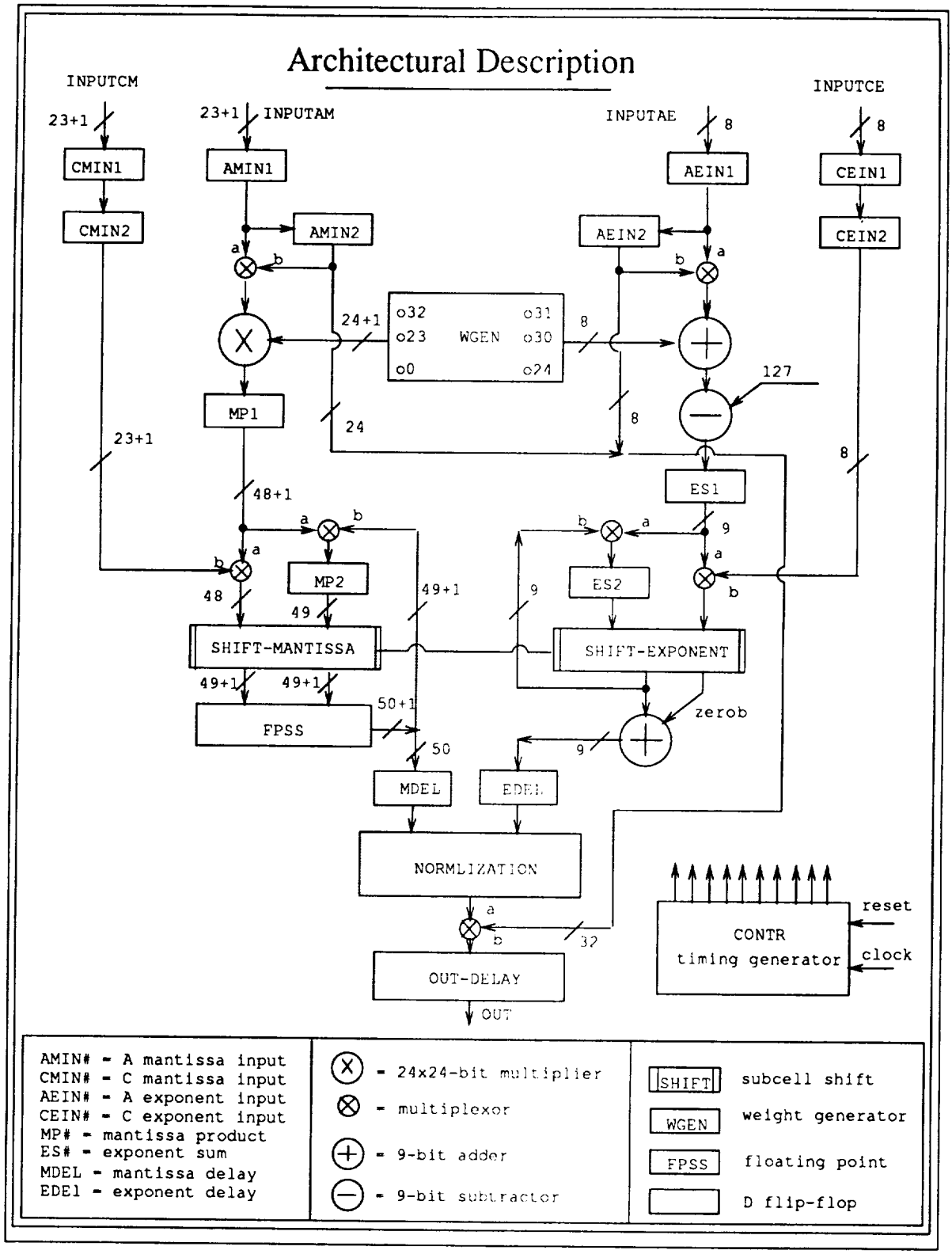


Figure 1: Architectural Description of SAFT64

2 The Subcells

Subcells described here are basic components in the SAFT64 chip. Some of them are repeatedly used in the chip layout. Since most data path widths are 32 bits, for a control signal, 32 driving loads are typical. Therefore, evenly distributing a drive signal is an important issue. All subcells have been optimized in the meaning of speed and silicon area. Those cells may be used in the future project as standard cells.

2.1 Edge Triggered D Flip-Flop

An edge triggered D flip-flop called ETD is the most common component [MAN84] in the chip.

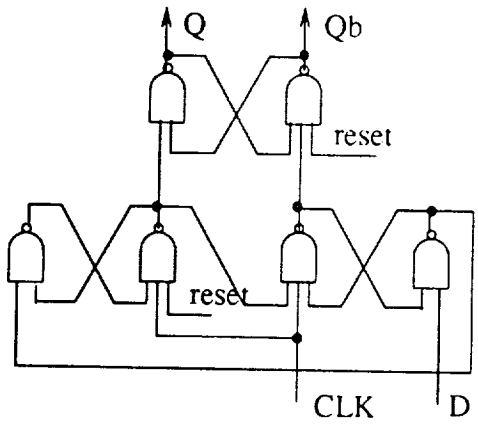


Figure 2: Logic diagram of the edge triggered D flip-flop

It is used as either a data latch or a time delay buffer. At the beginning

a level triggered D flip-flop was adopted. Since a level triggered D flip-flop may change its output status at any time as long as the clock is asserted, it makes chip timing difficult. But the level triggered D flip-flop occupied less silicon area than a edge triggered D flip-flop . Here, in our case, silicon area is not a very important issue because the chip uses a 79mm x 81mm frame, there is a plenty of space left for future use. Anyhow, the silicon area for the D flip-flop was minimized. The output of the D flip-flop will follow the input at the time a rising CLK edge is received. This protocol is defined by chip operation timing (see section 2.4 Chip Timing). Logic diagram for edge triggered D flip-flop is shown in Figure 2 and schematic of the 32ETD is in Figure 3. Every 8 flip-flops are driven by one big inverter which is, in turn, driven by a CLk signal. Since the CLK delay to each flip-flop is the same time, there is no clock skew problem here.

2.2 The Multiplexer

The MUXD is another frequently used subcell in the chip. It is a simple multiplexer. There are two inputs called A and B, one output O and one control select A, when the select A signal is high, the output of the multiplexer is following input A, otherwise, it follows input B. Logic diagram and schematic

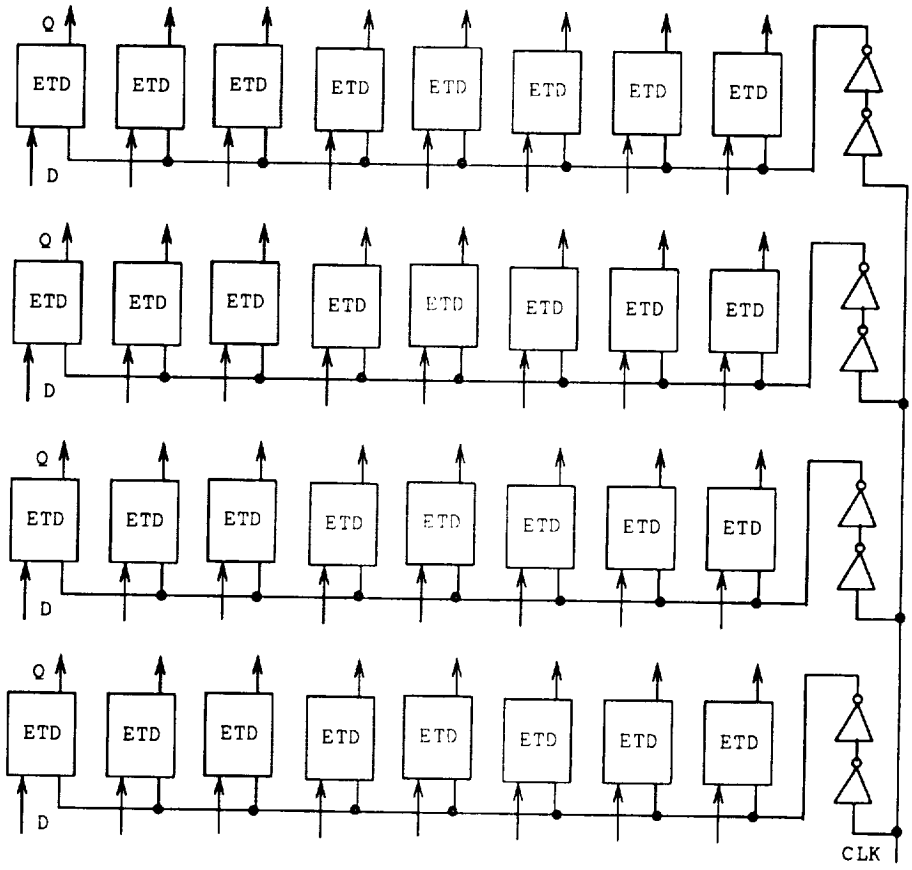


Figure 3: Block diagram of the 32ETD

of the MUXD are shown in Figure 4 and 5 respectively. A driver was added in the output level to improve the load capability. Simulation showed that if this driver was not used, the MUXD would be very slow, sometimes it would not work at all if the loads were too heavy. The same consideration was applied to the MUXD control signal as ETD, every 8 multiplexers are driven by a big inverter. According to the RSIM simulation, the delay time from applying 1 to the select A to getting the input A at output is 3.6ns.

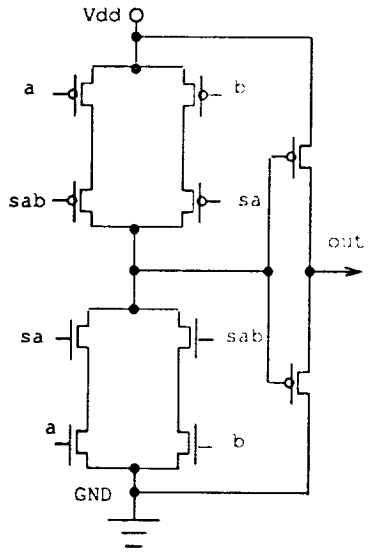


Figure 4: Logic diagram of the multiplexer

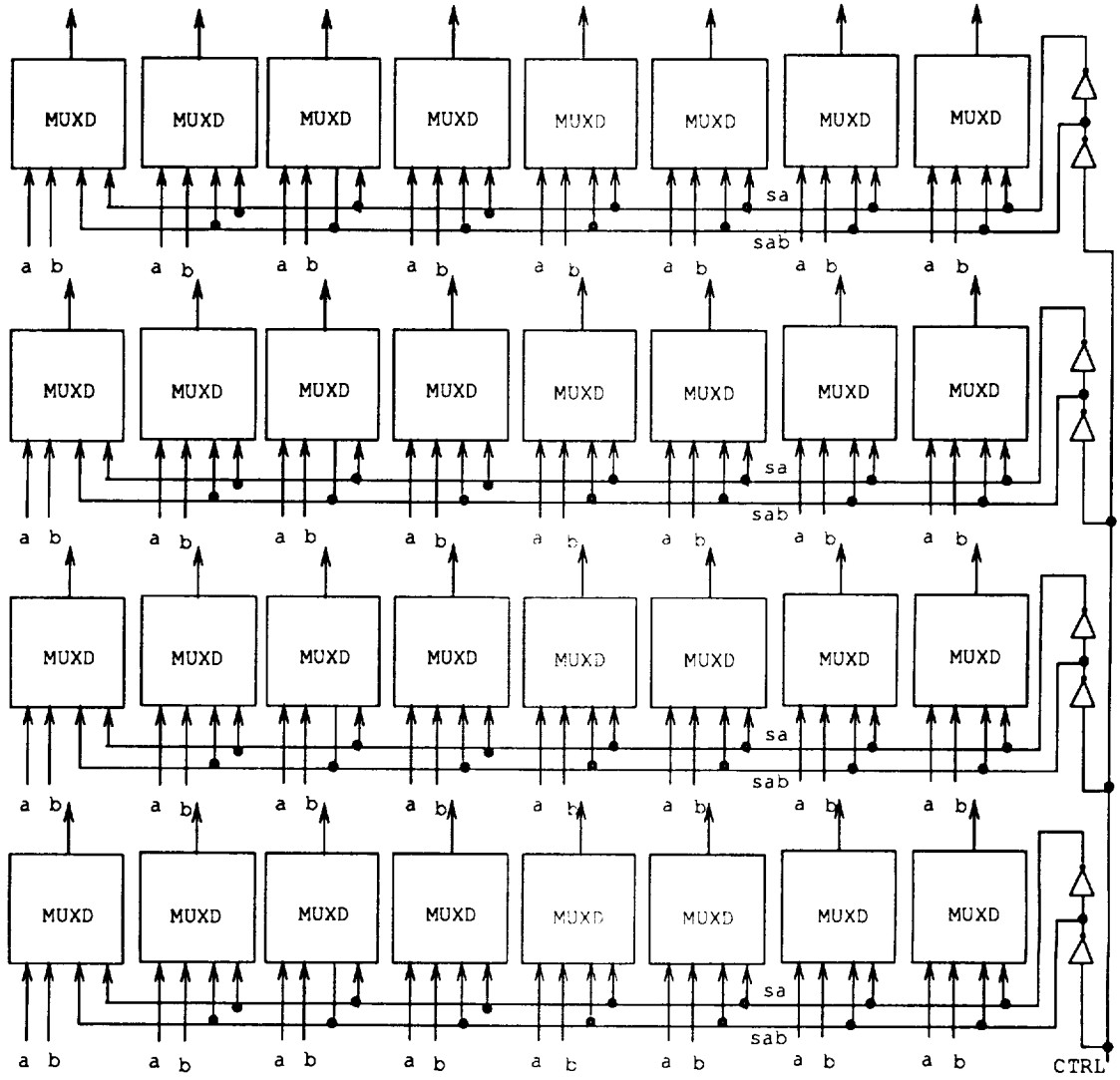


Figure 5: Block diagram of the 32MUXD

2.3 The 24x24-Bit Parallel Multiplier

This 24 bits multiplier is the biggest subcell in the chip. It has 24376 transistors. The design of the multiplier is based on Joseph Lee's work [JOE87] which was done in Hughes Research Laboratories in 1987. The advantage of this approach is that design issues are minimal, and the layout is highly modular. A logic diagram of the 24x24-bit parallel multiplier is shown in Figure 1.3.1.

As we can see from Figure 6, the entire multiplier consists of three subcells: a full adder, an AND gate, and a half adder. For simplicity, all the half adders were provided by connecting a carry input of the full adder to the ground level. For better speed performance the last stage in Figure 6 is a 24-bit carry look-ahead adder instead of a ripple carry adder. Every four pairs of the input bits constructs a carry group. Every four carry of such groups was sent to a carry generator. Thus, there are three levels of the carry generators, the first level was used for carry generation of four inputs, the second level was for sixteen inputs or caries of four groups, and the third level was for twenty four inputs. The final RSIM simulation shows that the maximum multiplication time for two 24-bit numbers is 135 ns.

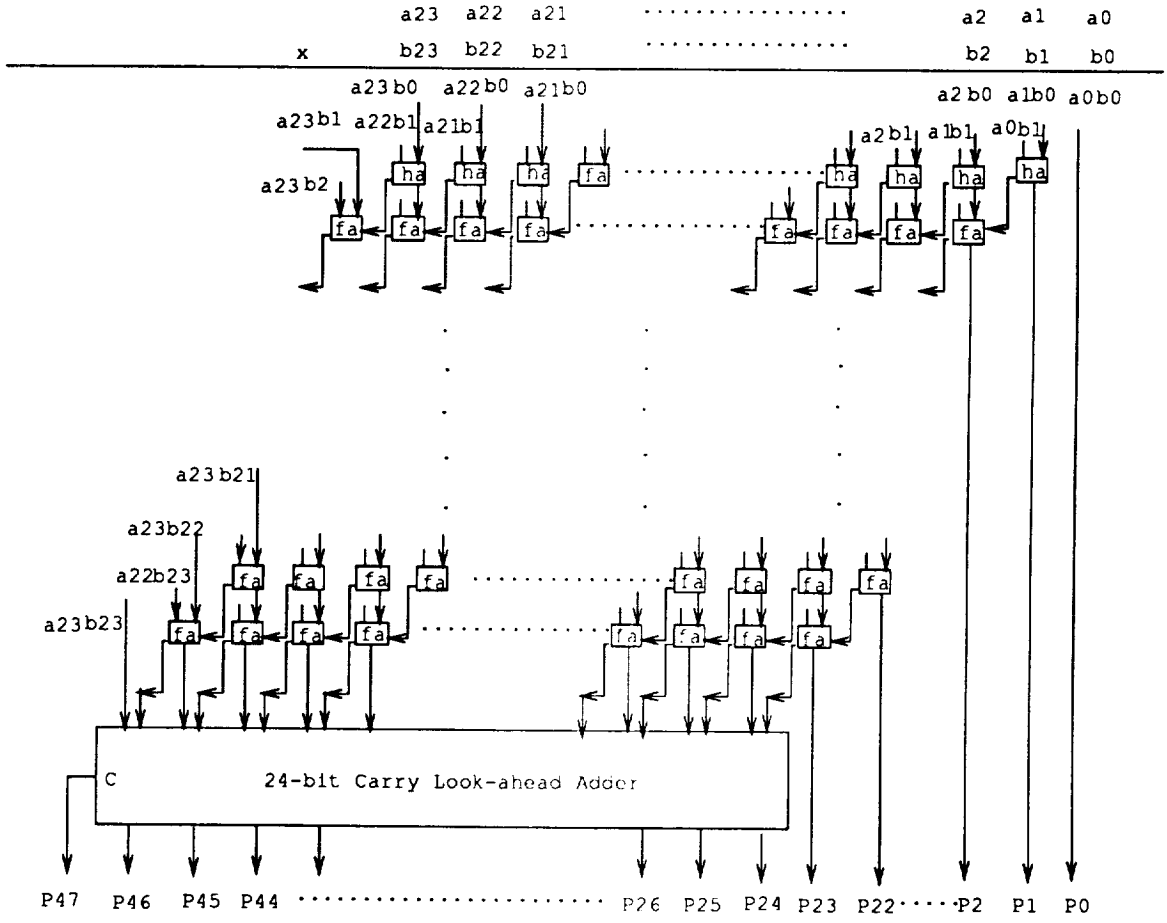


Figure 6: Logic diagram of the 24x24-bit multiplier

2.4 The Chip Timing Generator CONTR

Timing sequences of the internal operation were generated by a on chip timing generator called CONTR. A four-stage pipelined structure is implemented and controlled by those timing sequences. Furthermore, an inner-product function $X = A*B+C$ is calculated by such a structure. In subsection 2.4.1 the notation of control signals and their operational sequences are presented. Subsection 2.4.2 focuses on the detailed description of the implementation.

2.4.1 Notation of Control Signals

Figure 7 is a block symbol for the CONTR. There is only one input to the CONTR – an external clock. The outputs of the CONTR are A1, A2, C1, C2, T1, T2, T3, M1, M2, M3, M4, NEXT, and REAL. See Figure 1, Chip Architectural Description, for the meaning and position of those control signals.

Since data flow for the mantissa on the left side is exactly parallel to that of the right side for exponent, the control signals for both sides are identical. A1 is a clock signal for storage registers AMIN1 and AEIN1; A2 is for AMIN2 and AEIN2; C1 is a clock signal for storage registers CMIN1 and CEIN1; C2 is for CMIN2 and CEIN2; T1 is a clock signal for storage registers MP1 and ES1; T2 is for MP2 and ES2; T3 is a clock signal for time delay registers

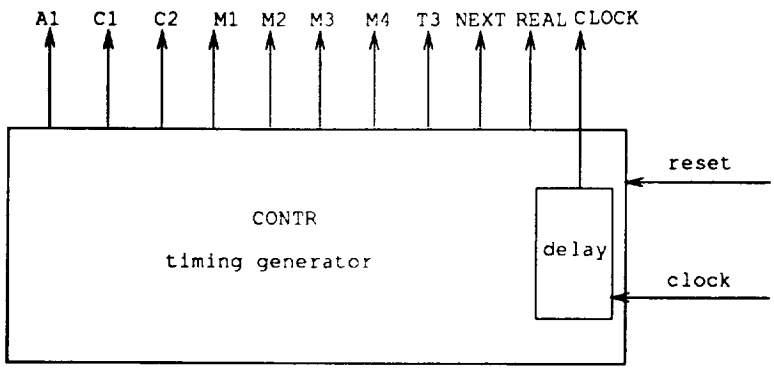


Figure 7: Block symbol of the CONTR

MDEL and EDEL; T4 is a clock signal for LDEL; M1 is a " select A " signal for multiplexers MIN_MUX and EIN_MUX; M2 is a " select A " signal for MP_MUX2

and ES_MUX2; M3 is also a " select A " signal for MP_MUX1 and ES_MUX1; M4 is a select A signal for OUT_MUX; NEXT is a clock signal for subcell the ROM; and REAL is a control signal for ROM's REAL. Here some special attention must be paid to NEXT and REAL. REAL is used to select a real weight or imaginary weight in ROM. When REAL is 1, a real weight is selected; otherwise, when REAL is 0, an imaginary weight is selected. The timing sequences were drawn from the graphic explanation of the data flow through the system [DEL90]. Some minor modification was made on his data flow chart, please see Section 3. From his data flow chart, it follows that a real weight or an imaginary weight must be repeated twice for a complex number

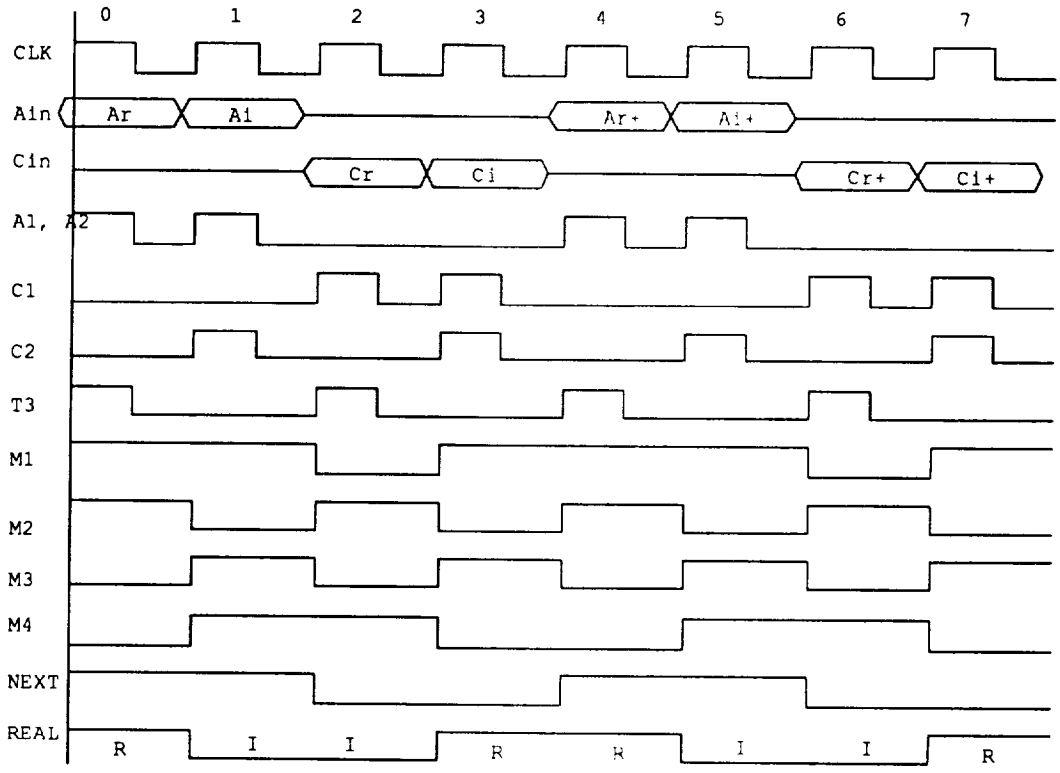


Figure 8: Internal operational timing

inner-product, i.e. at the first clock a real weight is output from the ROM, at the second clock the output of the ROM is changed to an imaginary weight, at the third clock the same imaginary weight must be kept on the output of ROM, and at the fourth clock the output of the ROM is changed back to the previous real weight. It appears that a memory location has to contain two weights, when REAL is 1, the output of the ROM is a real weight, while REAL is 0, the output is an imaginary weight of the same location. If there is a rising edge transition on the CK of ROM, the ROM will step to the next location. Therefore, the external clock cannot be directly connected to the CK of the ROM. This CK signal must be driven by a signal called NEXT which is generated by the CONTR. Through the data flow chart it is easy to see that T1, T2, and T4 are equal to CLOCK; and A1 is the same as A2. The CLOCK is an internal signal which has several gates delay time related to the external clock. This delay is artificially introduced to compensate for the delay generated by the 2-bit counter. More details are described in the next subsection. The chip timing sequences are plotted in Figure 8.

Those are the timing sequences necessary for internal operation. The timing sequences of input and output data as well as chip external control signals are described in detail in Section 5.

2.4.2 Detail Description of the CONTR Implementation

Basically the CONTR consists of two parts, a 2-bit counter and logic networks. Two edge triggered D flip-flops, two transmission gates and two inverters are employed to construct a 2-bit counter. Figure 9 is a logic diagram of the 2-bit counter. The 2-bit counter must be triggered at the rising edge of the clock signal. When RESET is high, the outputs of the counter are 00, independent of the changing of CLK. When RESET goes low, on each rising edge of CLK, the outputs of the counter are 01, 10, 11, and 00 respectively. Two drivers are added between the D flip-flops and transmission gates to increase the driving capability of D flip-flops. Other signals could be derived from the output of the 2-bit counter and the clock signal. Equations (1-13) show such relations among the output of the 2-bit counter and the clock signal.

The logic equations for the internal functions are :

$$A_1 = CLK * \overline{Q_1}; \quad (1)$$

$$A_2 = A_1; \quad (2)$$

$$C_1 = CLK * Q_1; \quad (3)$$

$$C_2 = CLK * Q_1; \quad (4)$$

$$T_1 = CLK; \quad (5)$$

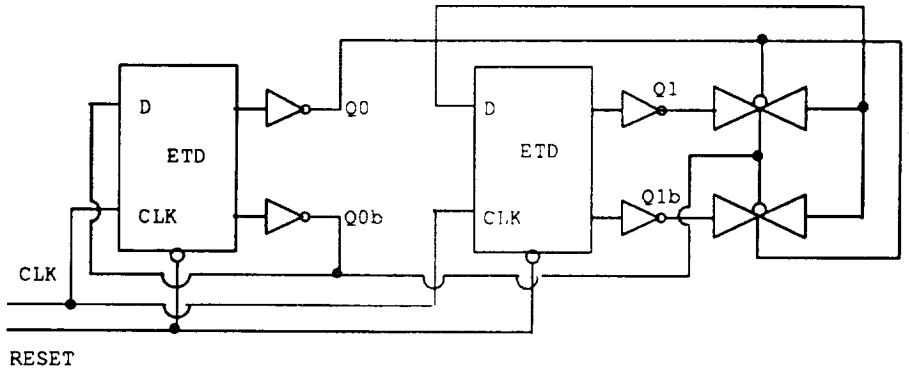


Figure 9: 2-bit counter

$$T_2 = CLK; \tag{6}$$

$$T_3 = CLK * \overline{Q_0}; \tag{7}$$

$$M_1 = Q_0 + \overline{Q_1}; \tag{8}$$

$$M_2 = \overline{Q_0}; \tag{9}$$

$$M_3 = Q_0; \tag{10}$$

$$M_4 = Q_0 exor Q_1; \tag{11}$$

$$NEXT = \overline{Q_1}; \tag{12}$$

$$REAL = \overline{M_4}. \tag{13}$$

Figure 10 shows a logic diagram of the CONTR. The upper part is the logic networks for the above control signals and the lower part is a 2-bit counter.

The delay unit depends on the time delay specified by the 2-bit counter. Ac-

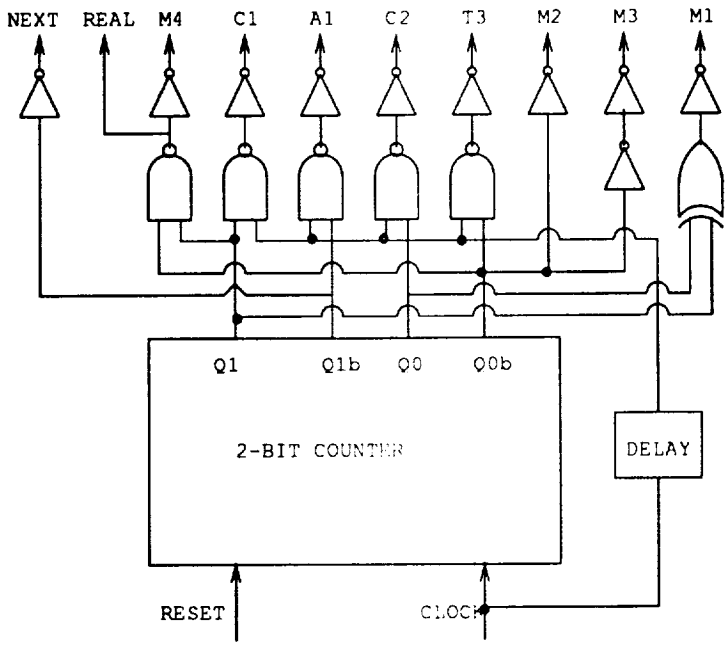


Figure 10: Logic diagram of the CONTR

tually, the 2-bit counter has a 8.4 ns delay . Therefore, the internal CLOCK is 8.4 ns later than the external clock CLK. Each control signal has been buffered so that bigger loads may be driven.

2.5 The Hidden Bit of the IEEE Floating Point Number

The IEEE floating point number format contains a hidden bit which is the most significant bit of the mantissa [IEEE85]. For convenience of reading, the IEEE floating point number format is redrawn in Figure 11.

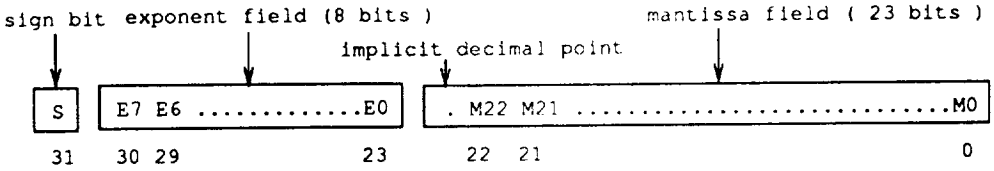


Figure 11: IEEE floating point number standard

As we can see, there are a 23-bit mantissa (b22...b0), a 8-bit exponent (b23...b30) and a sign bit (b31) for mantissa. The decimal point of the floating point number is between b23 and b22. For a normalized IEEE floating point number, the most significant bit must be 1, which is not represented in the IEEE floating point number format, and it must be generated by the hardware. The most significant bit of the mantissa is located to the left of the implicit decimal point. The exponent is offset by 127_{10}, such an exponent is called biased one. We assume that all floating numbers input to the SAFT64 are normalized to the IEEE floating point number. That means in most cases the hidden bit is a 1 except that the input number is a 0. When the input is a normalized zero, the exponent of this number must be zero. Therefore, a zero detector must be inserted in front of input latch AMIN1 and CMIN1. This zero detector is not drawn in Figure 1 for clarity in the

chip architecture. The function of the zero detector is quite simple, if 8-bit exponent is 0, then sets the hidden bit (MSB of mantissa) to zero, otherwise, it sets it to one. Figure 12 shows a logic diagram of the zero detector.

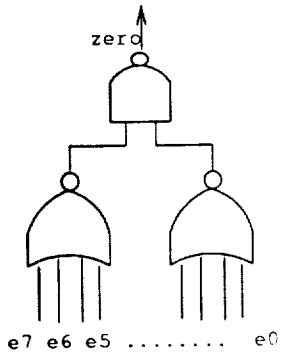


Figure 12: Logic diagram of the ZERO_DETECTOR

3 Modification of the Original Design

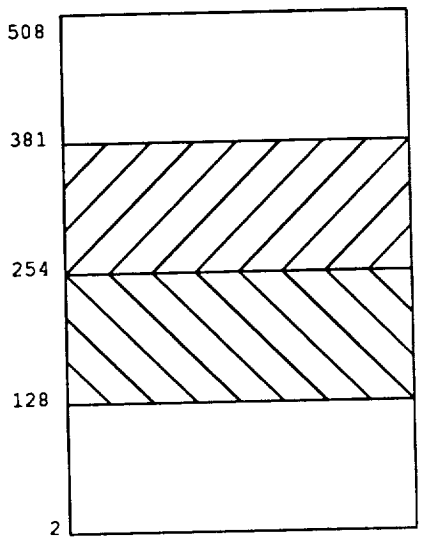
Some errors have been found in Delvecchio's design layout when the RSIM simulation is run for the ROM, the FPSS and the NORM subcells. The SHIFT subcell works perfectly. There were too many details to be considered at the design stage, some errors were be unavoidable. Those errors could only be found at the simulation of the integrated block.

3.1 The ROM

There were two obvious errors in the subcell ROM. The most significant bit of the ROM is "o32" which corresponds to the hidden bit of an IEEE floating-point number. It should be a 1 for a normalized IEEE floating-point number, and it should be a 0 if the value of the number is zero. In practice, the output of this bit is inverted. To correct this mistake, an inverter was added in the last stage of the bit o32.

The second error was the output of the exponent. As we know, the exponent of the IEEE standard floating-point number is biased, and this bias value (offset) for single precision is a decimal value of 127. the range of this exponent is from 1 to 254. When two IEEE floating-point numbers multiply, their exponents are added. The range of new exponent is from 2 to 508 which is beyond the IEEE exponent range 1 to 254. To fit this range into the IEEE exponent range any exponents which are less than 128 or greater than 381 must be eliminated, see Figure 13.

If the exponent of the ROM coefficients was not biased, we would be able to avoid such an exponent range mapping problem. Unfortunely, the true exponent output of the ROM was designed as biased one by Devecchio. To solve this problem, a subtractor is inserted after the two exponents are added. This subtractor always subtracts a decimal value of 127 from the exponent



In the IEEE floating point number standard

$e = 127 + E;$ $\quad 1 <= e <= 254;$

$-126 <= E <= 127.$

When two numbers multiply, their exponents are added, i.e.

$e1 + e2 = 127 + E1 + 127 + E2 = E_s$

To express this new exponent in a 8-bit word,

$128 <= E_s <= 381$, and

$E_n = E_s - 127.$

Figure 13: Overflow and underflow of two number's multiply

sum E_s and the result is called the normalized exponent E_n. Figure 14 shows a modified block symbol of the exponent adding unit. The subtractor actually is a 9-bit adder. One input of the 9-bit adder is E_s; another one is a 2's complement number of the decimal number 127. If the Carry of the adder is 1 (indicated a positive result), S_{0-8} are sent to the NORM; if the Carry is 0 (a negative and an underflow), $S_{8,7}$ are set to 11, at this time the exponent becomes an exponent that is greater than 256. The purpose of setting $S_{8,7}$ to 11 is because the NORM has only one exception handling capability; if the exponent is greater than 256, set the output exponent vector to 0 to indicate an overflow. Thus, if the normalized exponent is less than zero, then the subcell NORM will set this floating point number to zero. The reason for

this is quite straightforward; the number is too small to be presented by any other number except zero. If the normalized exponent is greater than 381, the output of the most two significant bits of the subtractor are automatically 11 which indicates an overflow. In turn, this nine bits exponent is fed to the NORM. The NORM should treat this exponent as an infinite or overflow, but for original design reasons it sets this floating-point number to zero.

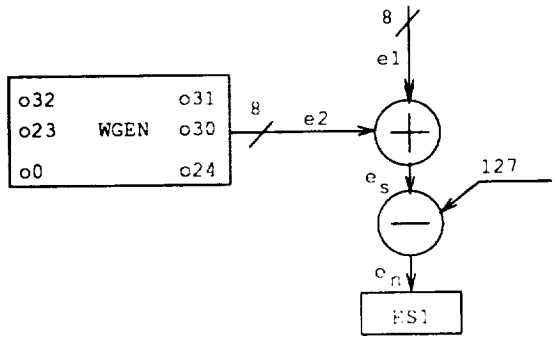


Figure 14: Logic diagram of the exponent handler

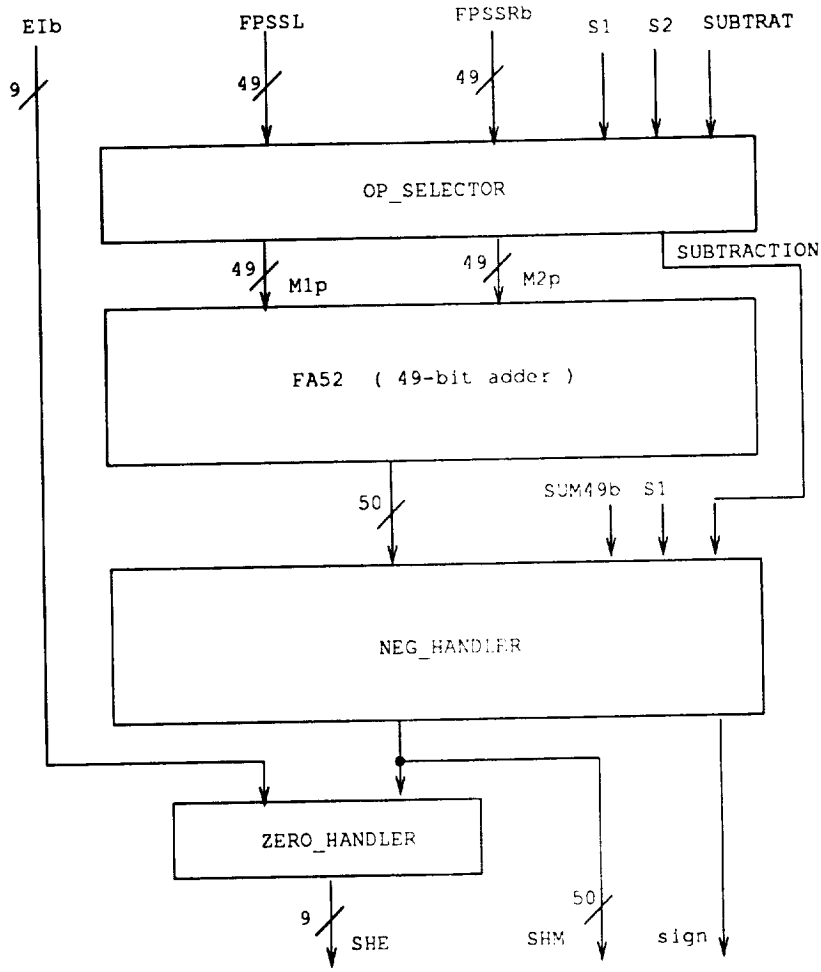
Another problem with the ROM is that the exponent stored was actual exponent plus three. This was resulted from the fact that a floating-point multiplication and an addition will shift the decimal point left at different stages. Since the decimal point is implied in the data, the shift operation is simply relative to the most significant bit of data instead of the decimal point. After $A * B$ has been carried out, the implied decimal point is on the left of the most significant bit. While $A * B + C$ is calculated, the implied decimal point of C is on the right of the most significant bit of C. If two

numbers with differently decimal point positions are to be added together, this fact leads to a computational error. For more details see Section 3.3 The NORM.

3.2 The FPSS

Figure 15 is a block symbol of the FPSS. Notice that one of inputs is inverted, FPSSRb, since this input was connected to the output of SHIFT and one of these output was inverted. This is a very important factor when design the OP_SELECT in the following section. The functions of FPSS are to: 1) convert inputs to the correct representation according to current operations - OP_SELECT; 2) add two numbers together - FA52; 3) change 2's complement number back to sign magnitude number if the result is negative - NEG_HANDLER; and 4) set the exponent of the floating point number to zero if the result (mantissa) is zero - ZERO_HANDLER.

Three points of the original design in FPSS have been modified. First, all control signals for multiplexers were changed from low effective to high effective; second, the outputs of adder FA52 are high effective; third, the logic of converting a negative result back to sign magnitude number in NEG_HANDLER has been changed. Some modifications shown in the fol-



S1 is sign of input A; S2 is sign of input B.

Figure 15: Block symbol for the FPSS

SUBTRACT	S2	S1	SUBTRACTION	OP_CTR2	OP_CTR1
0	0	0	0	0	1
0	0	1	1	0	0
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	1
1	1	0	0	0	1
1	1	1	1	0	0

Table 1: Truth table for the control signals of the OP_SELECT

lowing figures are just for logic simplicity. When two 49-bit numbers are added together the result may be a 50-bit number. Here in FPSS a subtraction $A - B$ ($A > 0$ and $B > 0$) is implemented by converting B (49 bits not including sign bit) into 2's complement number, and then added it with A. The sign bits of both A and B do not participate in the adding operation. The sign of the result is determined by bit49 SUM49, signs of A and B ($S1$ and $S2$), and what operation is being done (addition or subtraction). Table 1 and 2 illustrate such relations among the above logic variables.

As shown in Table 2 , when a true subtraction is carried out, the bit 49 of

SUBTRACTION	SUM49b	S1	SIGN	NEG_CTR
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	0	1
1	1	1	1	0

Table 2: Truth table for the NEG_HANDLER

the 49-bit adder FA52 determines the sign of result which is already in sign magnitude format. Therefore, even though a result is negative, only bit 0 to bit 48 need to be inverted.

Figure 16 is a logic implementation of Table 1. Two control signals OP_CTR1 and OP_CTR2 are used to control two exor vectors. When OP_CTR1 or 2 is logic zero, the outputs of the exor vectors remain the value of the inputs. When OP_CTR1 or 2 is logic one , the output of the exor 49-bit vectors might be opposite to the inputs. Initinally, a transmission gate was used instead of an exor gate, but it could not be driven by the SHIFT subcell. It then was replaced by an exor gate before the final simulation. Figure 17 shows how

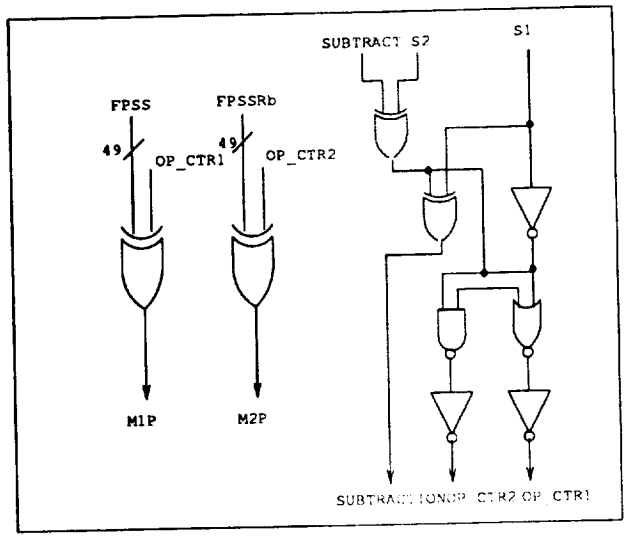


Figure 16: Schematic for the OP_SELECT

a sign is determined and when a data is converted back to sign magnitude representation. Here SUM49b is the most significant bit of the 49-bit adder FA52. If SUBTRACTION has been done by the FA52, the sign of the final result depends on SUM49. If SUM49 is zero, then the result is negative and needs to be converted back to the sign magnitude format; if SUM49 is one, no conversion needs to be done. If a additon (SUBTRACTION = 0) has been done, the sign of the final result depends on S_1.

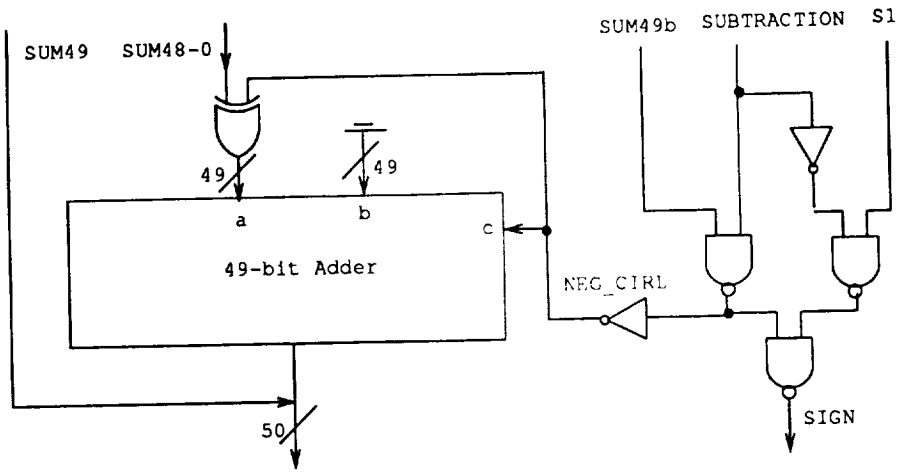


Figure 17: Schematic for the NEG_HANDLER

3.3 The NORM

As has been stated in the description of the IEEE floating point standard (Section 2.5), the implied decimal point is located between b22 and b23. In this section a more detailed discussion about the decimal point which is shifted after a multiplication and an addition is given.

Since a normalized IEEE floating point number is greater or equal to 1 and less than 2_{10}, the product of such two numbers is greater or equal to 1 and less than 4_{10}. The decimal point will be shifted as in the following binary representation:

$$1.b_{22}b_{21}b_{20}...b_1b_0 * 1.b_{22}b_{21}b_{20}...b_1b_0 = b_{47}b_{46}.b_{45}b_{44}b_{43}...b_1b_0.$$

Notice that the product is a 48-bit word because two multiplicands are 24-bit words. In order to calculate a complex number inner-product, four multiplications and four additions are needed:

$$R_{out} = A_r * W_r - A_i * W_i + C_r$$

$$I_{out} = A_r * W_i + A_i * W_r + C_i \ .$$

Let I_1 be $A_r * W_i$ and I_2 be $A_i * W_r$. Then,

$$I_1 + I_2 = b_{47}b_{46}.b_{45}b_{44}b_{43}...b_1b_0 + b_{47}b_{46}.b_{45}b_{44}b_{43}...b_1b_0 =$$

$$b_{48}b_{47}b_{46}.b_{45}b_{44}b_{43}...b_1b_0; \text{ and}$$

$$I_{out} = I_1 + I_2 + C_i = b_{49}b_{48}b_{47}b_{46}.b_{45}b_{44}b_{43}...b_1b_0.$$

Thus, the decimal point must be shifted left by three at the last stage. Now the problem was that in the internal stage the decimal point is not fixed, extra care must be taken by the designer. What DeVecchio did was that three was added to the exponent of the weight, it was equal to pre-shifting the decimal point left by three. In the internal stage any input number could be treated as if it had only one digit to the left of the decimal.

In practice, some difficulties arose. After adding three to the exponent, I_1 or I_2 become $x.xb_{47}b_{46}b_{45}...b_1b_0$, where $x.x$ did not exist in hardware,

but we treaded them as if they exist in the written representation. Thus, $I_1 + I_2 = x.b_{48}b_{47}b_{46}...b_1b_0 = I_3$. When $I_3 + C_i$, the decimal point should be in the same position, i.e.

$$x.b_{48}b_{47}b_{46}...b_1b_0$$

$$+ 1.b_{22}b_{21}...b_1b_0$$

then, the SHIFT will move the smaller one of I_3 and C_i right according to the difference of two exponents. Since the x bits do not exist, it is equivalent to having I_3 shifted left one more bit, thus generating an error. Therefore, a zero must be added in the most significant bit of I_3 to keep the decimal in the same position. Because the inputs of the SHIFT are two 49-bit ports, there is no more space for such an extra bit. Thus, I_3 or R_3 must be truncated off one more bit so that a zero was added as the most significant bit. This was accomplished by wiring bit 49 of I_3 or R_3 to bit 48 of the input port of SHIFT , and other bits were shifted one more bit to the right also. Notice that the bit 49 of I_3 or R_3 is always zero.

After a zero was inserted in I_3 or R_3, the calculation of I_3 or R_3 plus C_i or C_r was correct. But pre-shifting the decimal point three bits still has another problem, i.e. shifting the decimal point three bits reduced the dynamic ranges of additions. If there was no pre-shifting, I_3 or R_3 would have three digits to the left of the decimal point. Thus the addition of I_3 or R_3 plus C_i

or C_r would be

$$\begin{array}{lll} xxx.xxxxxxxxx...xxxx & \longleftarrow & I_3 \text{ or } R_3 \\ + \quad 001.xxxxxxx & \longleftarrow & C_i \text{ or } C_r \qquad (2.5.1) \\ \hline xxxx.xxxxxxxxx...xxxx & \longleftarrow & \text{final result } I \text{ or } R \end{array}$$

Then, three was added to the exponent to indicate that the decimal point was always shifted three bits. Obviously, the location of the decimal point would be fixed and easy to be found. If there were zeros to the right of the decimal point after adding three to the exponent, cell NORM would be able to handle it.

Let us now look at the case where pre-shifting has been done. I_3 or R_3 plus C_i or C_r becomes

$$\begin{array}{lll} 0.xxxxxxxxx...xxxx & \longleftarrow & I_3 \text{ or } R_3 \\ + \quad 1.xxxxxxx & \longleftarrow & C_i \text{ or } C_r \qquad (2.5.2) \\ \hline xx.xxxxxxxxx...xxxx & \longleftarrow & \text{final result } I \text{ or } R \end{array}$$

There were two digits to the left of the decimal point, which meant that to convert the final result to the normalized IEEE floating point number, there was one more bit to be shifted to the right though we had concluded that the decimal point must be fixed in the position one digit left in any

internal stages. In other words, the final result was bigger than we expected. Comparing calculation 2.5.1 with 2.5.2, we could see that pre-shift decreased the dynamic range of additions. In calculation 2.5.1 there were three digits to the left of the decimal point, to put C_i 's decimal point in the same position two zeros were implicitly inserted, which ensured that the final result was less than 10_{10} or four digit to the left of the decimal point (see DeVecchio's thesis, Section 2.7). In calculation 2.5.2, there was only one digit to the left of the decimal point and the most significant bit of C_i was always one, a carry may be generated, which led to the fact that the final result was bigger than it should be.

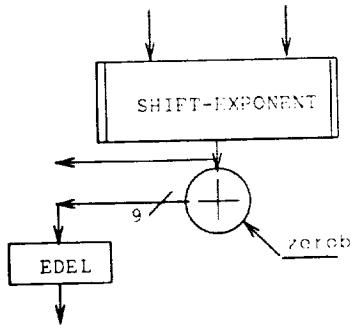


Figure 18: Logic diagram for the modified exponent

To solve this problem, a 1 was added on the exponent before it was fed to NORM as shown in Figure 18. As we had accepted the pre-shifting decimal point, there should be no trouble in accepting a post-shifting of the decimal point. In reality, a 9-bit adder was used between FPSS and NORM, and

adding one or not adding it depended on the ZERO_HANDLER in the FPSS subcell. If the output of the ZERO_HANDLER was not zero, a 1 was added to the exponent; otherwise, a 0 was added.

4 Simulation

The simulation was done using RSIM. Since a four-stage pipeline structure was employed, it made the simulation more difficult. The only way we could do it was to put simulation labels at the outputs of each component to track what was going on. Each label represented a test point. At each point we see a vector consisting of several bit components. We use a bit to express a particular component of a vector in the following paragraph. The labels used in the simulation are shown in Figure 19.

There are four input vectors, exponent of input A - INPUTAE, mantissa of input A - INPUTAM, exponent of input C- INPUTCE, and mantissa of input C - INPUTCM. INPUTAE is a 8-bit vector which is used to express an exponent of an IEEE floating point number. INPUTAM is a 24-bit vector which containes the sign bit of an IEEE floating point number in bit INPUTAM23 and mantissa from bits INPUTAM22_0. The same protocol is applied to INPUTC. INPUTAM and INPUTAE are called as input port A,

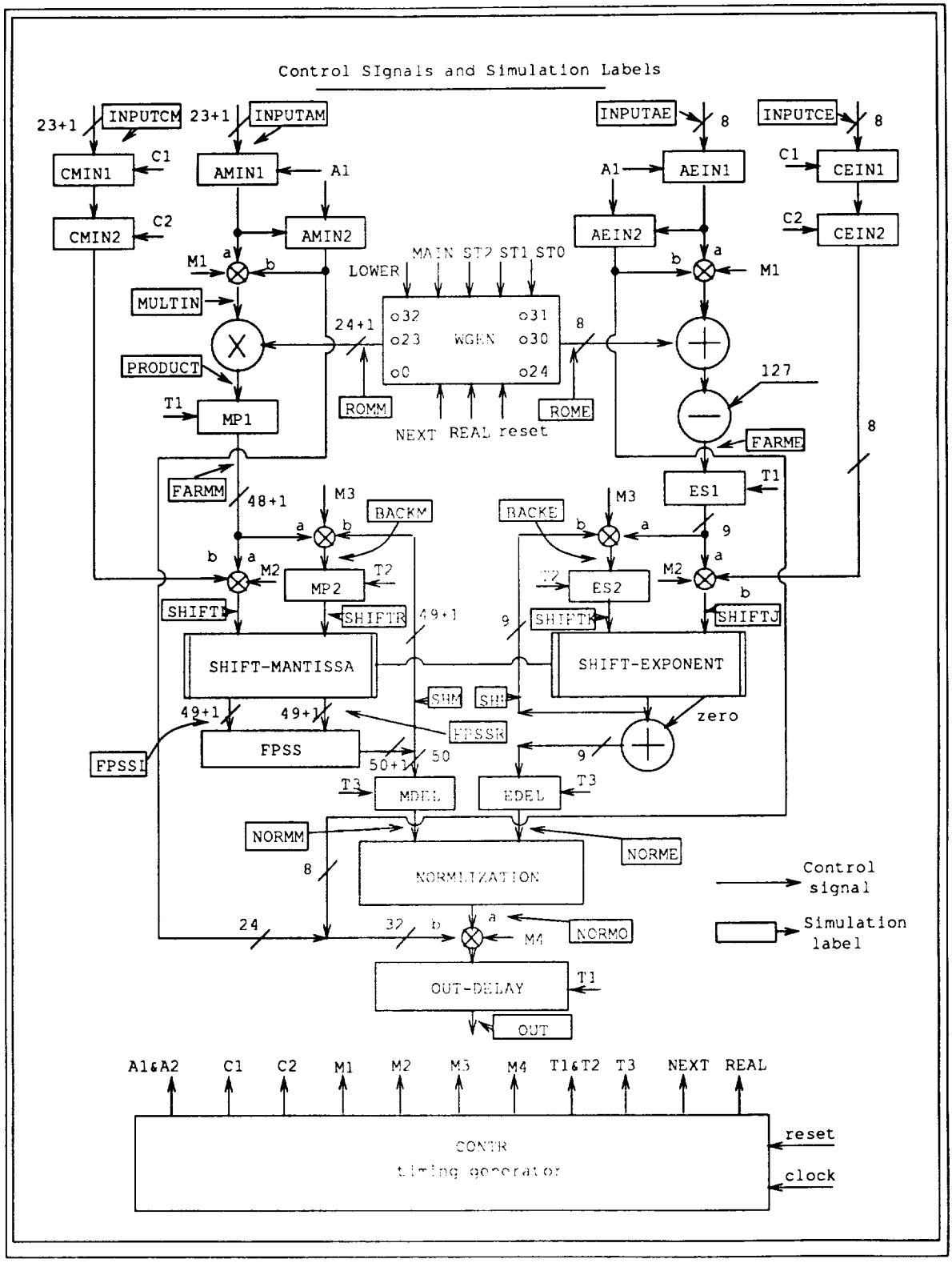


Figure 19: Control signals and simulation labels

CLK	IN	MUL	ROME ROMM	SUM PRO	FAE FAM	BAE BAM	SHJ SHR	SHK SHL	SHE SHM	NORE NORM	OUT
0	A_r	A_r'	W_r	R_1	X	X	X	X	X	X	X
1	A_i	A_i'	W_i	R_2	R_1	R_1	X	X	X	X	X
2	C_r	A_r'	W_i	I_1	R_2	R_3	R_1	R_2	R_3	X	X
3	C_i	C_i'	W_r	I_2	I_1	I_1	R_3^-	C_r	R_4 X	X	X
4	$A_r{}^1$	$A_r{}^{1'}$	$W_r{}^1$	$R_1{}^1$	I_2	I_3	I_1	I_2	I_3	R	A_r
5	$A_i{}^1$	$A_i{}^{1'}$	$W_r{}^1$	$R_2{}^1$	$R_1{}^1$	$R_1{}^1$	I_3	C_i	I_4	R	A_i
6	$C_r{}^1$	$C_r{}^{1'}$	$W_r{}^1$	$I_1{}^1$	$R_2{}^1$	$R_3{}^1$	$R_1{}^1$	$R_2{}^1$	$R_3{}^1$	I	R'
7	$C_i{}^1$	$A_i{}^{1'}$	$W_r{}^1$	$I_2{}^1$	$I_1{}^1$	$I_1{}^1$	$R_3{}^1$	$C_r{}^1$	$R_4{}^1$	I	I'

Table 3: Another form of the data flow chart

and INPUTCM and INPUTCE are called as input port C. Since both real and imaginary numbers are input through port A and C, we use A_r and A_i to indicate the real part of A and the imaginary part of A. The same rule was applied to C. Table 3 is another form of the data flow chart, where x indicated a "don't care" item, a superscript number indicated a sequence of the input data, and superscript ' indicated a normalized item.

5 The Signal Protocol

In this section some basic timing for external signals are introduced, and those timing relations are derived from the RSIM simulation, they may change slightly after the chips are tested. Another part of this section is

the pinout of the SAFT64. There are total 132 pins on the chip, of which 112 pins have been used, 20 pins are left unused for future extensions or chip test pins. The timing described here does not include differential pat delays.

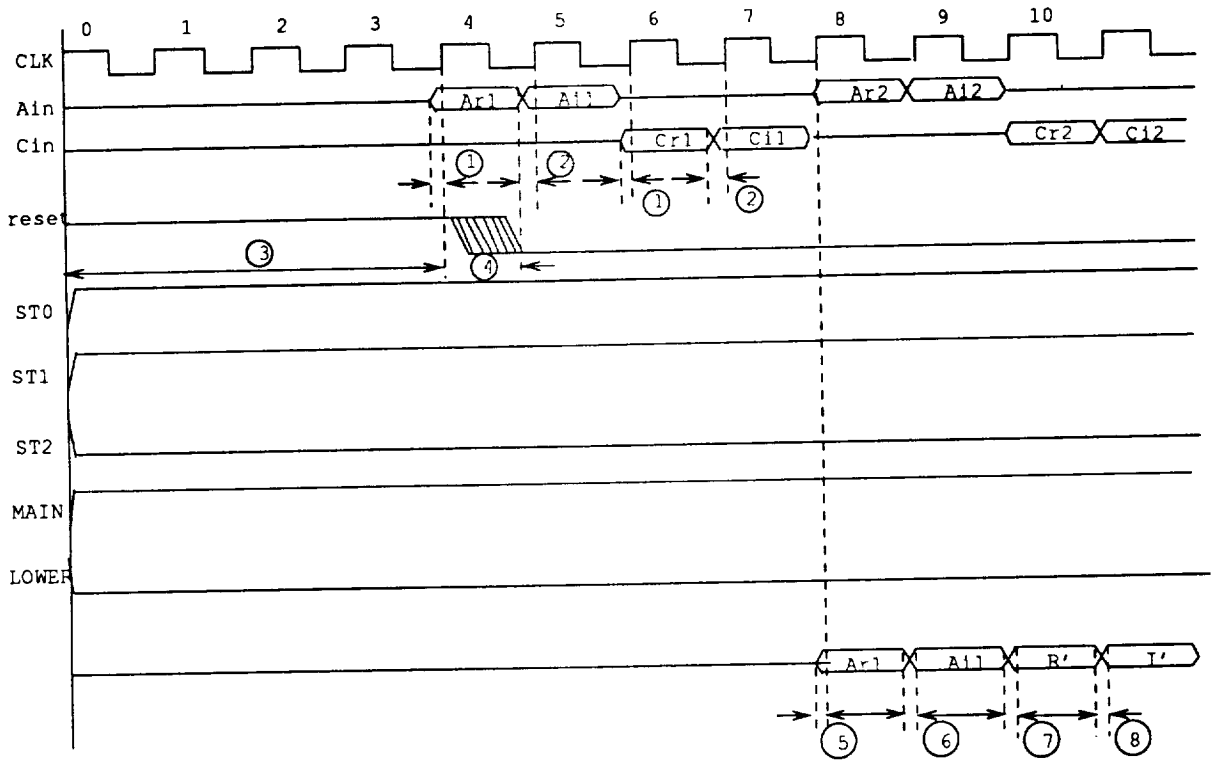
5.1 Operational Signals

There are eight signals which control the operations of the chip, and there are three data ports on the chip.

1. CLOCK (input): this signal is provided by the users. The typical clock period is 250ns with duty cycle from 30% to 70%. Simulation showed proper operation for clock periods from 200ns to 400ns.

2. RESETI (input): this signal sets the weight generator WGEN to the first memory location and the timing generator CONTR to start state. It must be applied at least four clock cycles before the first valid data word arrives at the input port A or C and it has to be held high until the first data is stable at the data port, i.e. it may be turned off 4 ns after the first data is applied. It should be dropped before the second positive rise of the CLOCK.

3. RESETO (output): an output signal to the next chip. It is just a four-clock cycle delay of the RESETI. When many chips are cascaded together, the external signal Reset goes to the first chip, and the RESETO of the first



① Minimum valid data time is 4 ns

② Previous data off time is 6 ns

③ High for at least 4 clock periods

④ The time of Reset off must be in the interval of the first data

⑤ The input appears at output after 4 clock cycles it is input to the chip, and this time is the same as ① .

⑥ Same as ②

⑦ Same as ①

⑧ Same as ②

Here CLK is an internal clock which is 8.2 ns later than external clock. Other signals must be kept constant durig the operation. The above ia an example, thestep size is 4, and the main diagonal weight is selected.

Figure 20: I/O timing

ST2	ST1	ST0	STEP SIZE
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	undefined
1	1	1	undefined

Table 4: Truth table of the step size

chip will be connected to the RESETI of the second chip, etc, (See Figure 22.

4. Step size ST0, ST1 and ST2 (input) : the three control lines identify the position of the chip in the processor array and move the memory pointer of the weight generator to the proper location. Table 4 shows the functions of those signals. ST0, ST1 and ST2 must be connected to GND or Vdd during chip operation.

5. Weight Diagonal specification MAIN and LOWER (input): the combination of those two signals is to determine which diagonal of the weight ROM is used in the current calculation. There are three diagonals in the

MAIN	LOWER	DIAGONAL
1	0	main
0	1	lower
1	1	upper
0	0	undefined

Table 5: Truth table of the diagonal selection

ROM, the MAIN, the LOWER and the UPPER. Table 5 shows how a diagonal is selected by those two signals. MAIN and LOWER must be connected to GND or Vdd during the chip operation.

6. Data Port A (input): is a 32-bit data input port for digital signals. Here the so-called digital signal means the digitized data obtained from A/D sampling and some filtering real and imaginary words in sequence.

7. Data Port C (input): this 32-bit data port accepts the data generated by the next SAFT64 chip (See Section 6).

8. Data Port O (output): outputs the results of calculating $A * W + C$ by the chip under consideration.

We assume that all data input to port A and C are normalized IEEE floating point numbers. The output of the port O is a normalized IEEE floating

point number also. Notice that the SAFT64 does not fully support the IEEE floating point standard; NAN and positive and negative Infinite are not included in the possible outputs of the Port O. The input data must be valid 4ns before each clock rise, see Figure 20.

Data input sequence: the first data word, A_r, must be a real number and the second data word, A_i, must be an imaginary number, both A_r and A_i are the real and imaginary parts of the first input data point. The C_r input must be applied before the next positive clock transition and C_i must follow C_r at the next clock transition. All of these data words must be present for one clock cycle and must be valid 4ns before the rising edge of the clock. New A_r and A_i values can follow C_i immediately. After the input data sequence (64 complex words) is complete a new sequence can start immediately.

Data output sequence: after A_r is applied four clock positive transitions, A_r appears at the output without modification. Here we count the first clock transition as the one that took A_r into the chip. After A_r appears, it is followed by A_i one clock period later. Yet another clock period later the result of the computation real number R appears at the output. One clock later I is presented at the output.

5.2 The Chip Pinout

There are total 132 pins on the chip, of which 112 pins are used. Pin 1, Pin 67, Pin 101 and Pin 112 are ground GND; Pin 34, Pin 100, Pin 110 and Pin 124 are positive 5 volt power supply Vdd. Port A occupies Pin 2 to Pin 33; Port C uses Pin 43 to Pin 66 and Pin 125 to Pin 132; and Port O carries Pin 68 to 99; Pin 35 to 42 are control signals; and Pin 102 to 127 are test pins which are not used by the users. Figure 21 shows this configuration.

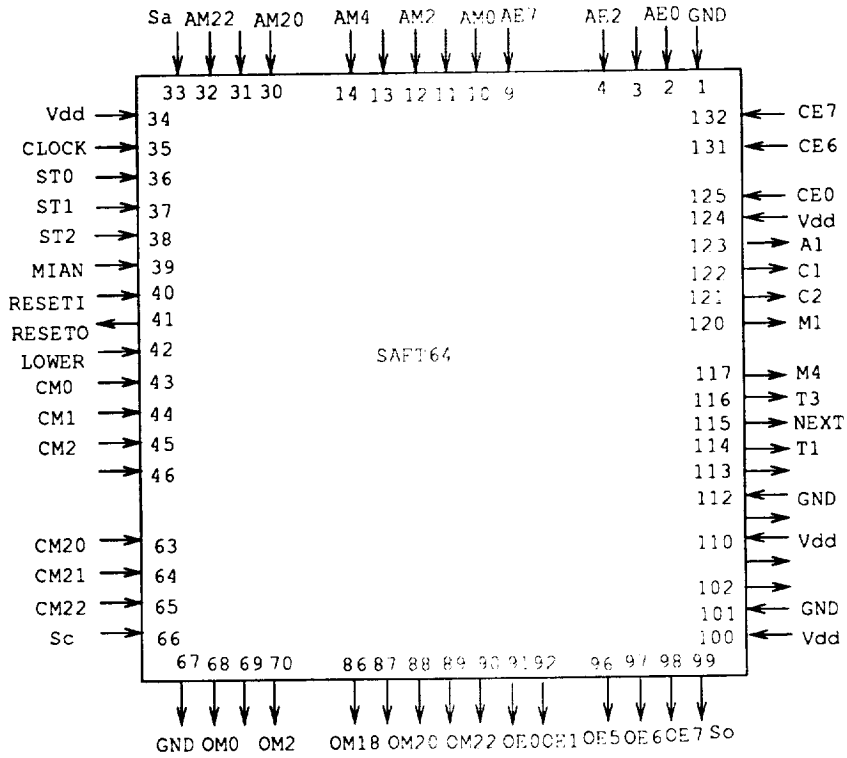


Figure 21: The chip's pinout

6 Applications of the SAFT64

A configuration using SAFT64 chips to implement a 64-point FFT is shown in Figure 22. Eighteen SAFT64 chips are used. All control signals have been wired to proper terminals. Since the chips in different levels have different delay time which has been fulfilled by internal delay registers, and the delay time of those registers is determined by ST0, ST1 and ST2. Therefore, connection of the chips is quiet straightforward and simple.

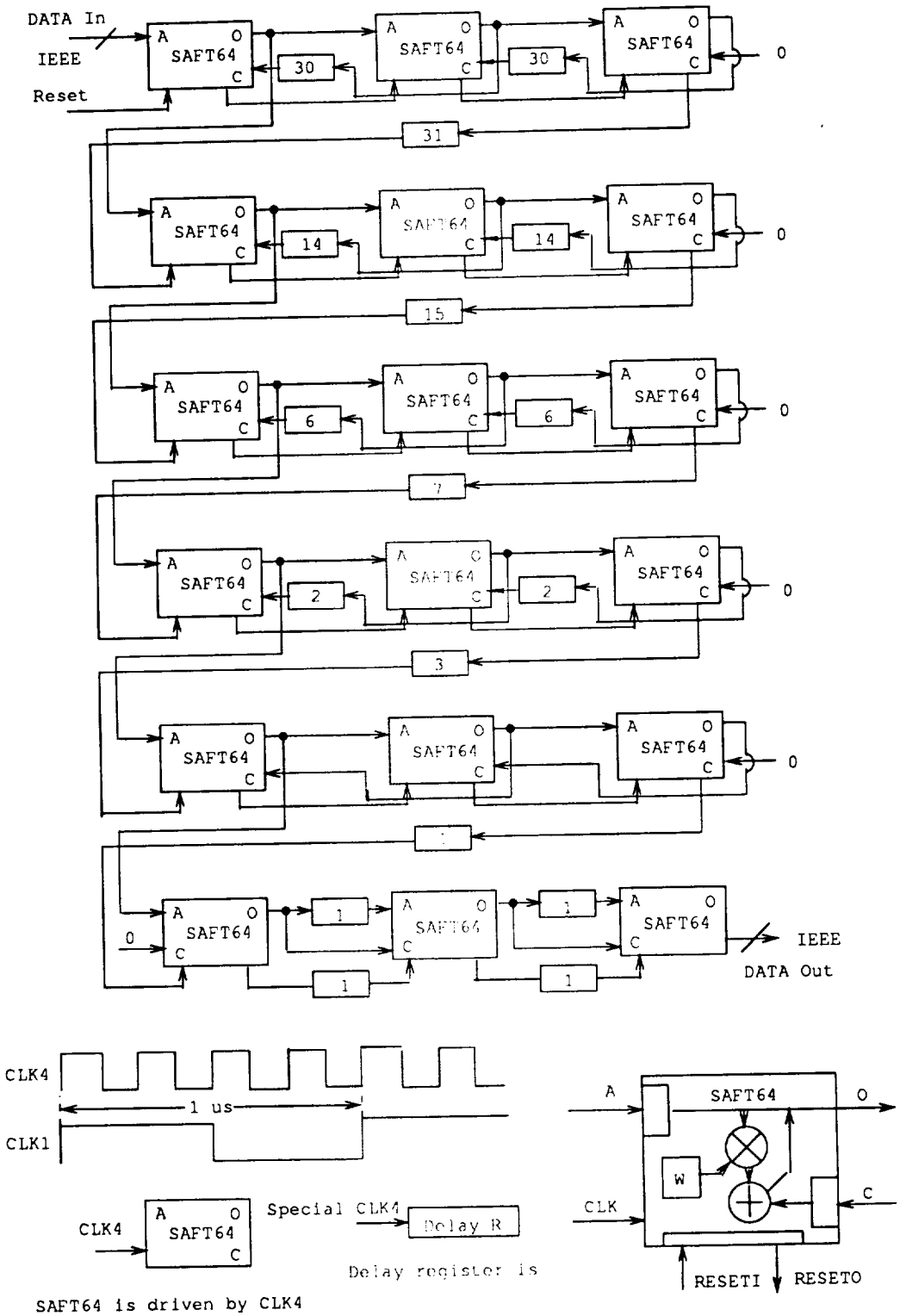


Figure 22: Configuration of using SAFT64 to do 64-point FFT

References

[BOR88] Boriakoff, V.,"FFT Computation with Systolic Arrays, a New Architecture", Submitted for publication by V. Boriakoff, Worcester Polytechnic Institute, 1988.

[MAN84] Mano, M. Morris *Digital Design, Printice Hall, New Jersey, 1984.*

[JOE87] Kamark, Joefery, " A High Density 8x8 Parallel Multiplier", IEEE transctions on Circuit and System, vol , 1988.

[DEL90] DelVecchio, P. E. *The Design and Formal Verifcation of an Integrated Circuit for Use in a Floating-Point Systolic Array Fast Fourier Transform Processor, Master Thesis, Cornell University, Ithaca, New York, 1990.*

[IEEE85] IEEE Standard for Binary Floating-Point Arithmetic, (ANSI/IEEE std 754-1985), New York: The Institute of Electrical and Electronics Engineers.

Appendix

The following is a simulation file which was produced by the RSIM simulation. All simulation labels n the file are coresponding to the labels in Table 3. For example, a MUL in Table 3 corespones to "multin" in the simulation file; a FAM in the table means "farmm" in the file. Since the width of the paper is not wide enough to hold so many simulation labels, abbreviations of those labels have to be used in Table 3.